DOCUMENT RESUME

ED 305 920 IR 013 780

AUTHOR Yoder, Sharon Burrowes
TITLE Teaching Basic and Pascal "Logo Style."
PUB DATE Jun 88
NOTE 15p.; Paper presented at the National Educational Computer Conference (Dallas, TX, June 15-17, 1988).
PUB TYPE Guides - Classroom Use - Guides (For Teachers) (052) -- Viewpoints (120)

EDRS PRICE MF01/PC01 Plus Postage.
DESCRIPTORS *Assignments; *Creativity; *Discovery Learning; *Evaluation Criteria; High Schools; *Programing; Programing Languages; *Student Evaluation; Teaching Methods
IDENTIFIERS *LOGO Programing Language

ABSTRACT

In traditional programming courses assignments usually consist of problems that are somewhat closed in nature. That is, students are usually given a rigidly defined problem statement and are provided with a set of data that they are to use to test their program to produce a unique result. While this approach makes evaluation easy, it discourages creativity and exploration, and encourages inappropriate sharing of information. Another approach involves open-ended assignments that encourage students to make the problem their own and to include their own creative ideas in their work. Such projects require that the teacher carefully examine the objectives for a particular assignment and plan the evaluation scheme thoughtfully. However, student response to such open-ended assignments is quite positive and often very creative. An important side effect of such assignments is that group interactions are much more appropriate. Students help debug each others' projects rather than simply trading algorithms. Further, students learn from each person with whom they interact, thus extending their knowledge well beyond that acquired when solving a single solution problem. (4 references) (Author)

****
* Reproductions supplied by EDRS are the best that can be made *
* from the original document. *
****

ED305920

U.S. DEPARTMENT OF EDUCATION
Office of Educational Research and Improvement
EDUCATIONAL RESOURCES INFORMATION CENTER (ERIC)

☒ This document has been reproduced as received from the person or organization originating it.

☐ Minor changes have been made to improve reproduction quality.

• Points of view or opinions stated in this document do not necessarily represent official OERI position or policy.

Teaching Basic and Pascal "Logo Style"

Sharon Burrowes Yoder
Logo Computer Systems, Inc.
330 West 58th Street, Suite 5M
New York, New York 10019

Audio/visuals needed for the presentation:
Apple IIc, 128K IIe, or II GS
Large screen display, color preferred
Overhead projector

IR013780

"PERMISSION TO REPRODUCE THIS MATERIAL HAS BEEN GRANTED BY
Sharon Yoder

TO THE EDUCATIONAL RESOURCES INFORMATION CENTER (ERIC)."

Abstract

In traditional programming courses assignments usually consist of problems that are somewhat closed in nature. That is, students are usually given a rigidly defined problem statement and are provided with a set of data that they are to use to test their program which will produce a unique result. While this approach makes evaluation easy, it discourages creativity and exploration, and encourages inappropriate sharing of information.

Another approach to assigning programming problems is to use open-ended assignments that encourage students to make the problem their own and include their own creative ideas in their work. Such projects require that the teacher examine carefully the objectives for a particular assignment and plan the evaluation scheme thoughtfully.

Student response to such open ended assignments is quite positive and the resultant work is extremely varied and often very creative. An important side effect of such assignments is that group interactions are much more appropriate. Students help debug each other's projects rather than simply trading algorithms. Further, students learn from each person with whom they interact, thus extending their knowledge well beyond that acquired when solving a single solution problem.

Introduction

The core of most computer programming instruction, whether it be embedded in the elementary curriculum, part of a junior high module, or semester or year long high school or university courses, is the hands-on assignments that require students to make use of the programming language that they are learning in the classroom. Assignments for such courses tend to be carefully specified, usually requiring that the finished product behave in a rigid manner, often using a specific set of data.

This approach, while it does require students to make use of the computer to solve problems, has several weaknesses. First, it allows for little or no creativity or exploratory learning. While the students may choose somewhat different algorithms for solving such problems, there are usually a limited number of methods of solution, especially with problems appropriate to beginning programmers. Second, all too often only a few of the best students actually solve the problem assigned; the remaining students merely adapt the algorithm developed by the few. Sometimes this is conscious cheating on the part of students; at other times, those seeking help simply end up getting too much assistance. In either case, the often implicit goal of teaching problem solving is quickly lost for the majority of students.

The Logo philosophy offers an alternative way to think about teaching programming. Ideally, students in a "Logo environment" are encouraged to develop their own projects and to use exploratory learning and discovery to reach their goal. It is possible to merge this open ended, discovery approach to the learning of programming

with the traditional approach to teaching any programming language. This can be done in such a way that students become more engaged in what they are doing while maintaining standards of learning that are typically required in a programming course.

The techniques for the teaching of programming to be described in this paper were developed over a several year period in a semester Introduction to Computers and Computing course offered to students in grades 9 - 12 in a small-town high school in the Midwest. These ideas were more recently applied to other high school courses including courses in Basic and Pascal, Advanced Placement Computer Science and to teacher training courses. While the fundamental technique has remained unchanged, problem sets and evaluation schemes continue to evolve as they are used with more students and in a variety of courses.

The ideas for giving open-ended assignments grew out of the Logo philosophy, put forth most clearly by Seymour Papert in *Mindstorms: Children, Computers, and Powerful Ideas.* The "Logo environment" encourages open-ended projects and exploratory learning. Students involved in such self-directed projects become more deeply engaged in their work because their goals are their own; rather than being imposed by an outsider.

A few years ago, the character of the students enrolled in the introductory high school computer course taught by this author changed from primarily "hackers" to students interested in a more general introduction to computers. Traditional rigid problems designed to prepare students for college-level computer science courses seemed less and less appropriate Initially the techniques

described in this paper were used with an occasional assignment. As the techniques became increasingly successful with students in a limited context, they were tried with a wider range of assignments and then with increasingly advanced students as well as inservice teachers, all with equal success.

Background

Anyone who has taken a traditional programming course has probably encountered a problem such as this:

> Write a program to compute the take-home pay for an employee of the Acme Computer company, given the hourly rate of pay and the number of hours worked. Any hours over 40 should receive time and a half. Run your program three times, once for each of the following sets of data: $5.25 for 23 hours; $9.95 for 51 hours; $7.53 for 40 hours.

Once a student has the necessary assignment statement in hand, whether it is acquired through individual problem solving techniques or from a friend, "solving" the problem becomes trivial.

Beyond the process of creating a program that runs, the instructor assigning such a problem no doubt has clear objectives in mind. Likely, in the case of the problem above, the objective is to have students use the assignment statement, conditionals, and arithmetic operators.

A Pascal textbook that has been used at both the high school and university level provides another typical problem:

> The input consists of groups of three numbers. The last group is an end-of-list signal consisting of three zero

> values. Write a program to read each group in, print it out, and print an indication of whether the three numbers represent the sides of a triangle. (Conway, Gries,and Zimmerman, p. 85)

What is the objective implied by this problem? Since it follows a chapter on loop structures, most certainly the authors expect the student to use either the Repeat-Until or While-Do construct. Further, it is clearly asking students to deal with an end of data condition that is not trivial.

From a more recent Pascal textbook comes a problem that acknowledges the sameness of so many programming problems:

> In consideration of all the business programs that compute retail prices, write a consumer's program that figures out wholesale prices. Given a purchase price, it should deduct a 40% markup, and a 5.5% sales tax. Be sure to make the deductions in the right order. (Cooper and Clancey, p. 67)

It appears that the authors of this text were uneasy with the traditional programming assignments, but did not completely break away from the mold, even though this book contains many problems that are much more interesting and challenging than those found in most programming books.

Clearly, the programming problems described here require the student to use specific aspects of the language being learned. The instructor can easily evaluate whether the program is correct or not by having the student run the program on a specific data set. Matters of program design and style can be dealt with by examining the program listing. Evaluation is relatively unambiguous and there is little that is subjective in the grading process.

Methodology

The first step in moving away from traditional programming assignments is the creation of open-ended problems that encourage discovery learning. This is perhaps the most difficult part of the teacher's task. It is important to be very clear about the objectives to be met before designing each assignment. An example of such an assignment is

> Write a program that uses music and graphics or music and text. Your program should have music playing while some action takes place on the screen. Be careful when designing your program so that the subdivision of your procedures make sense. Extra credit will be given for using music, graphics, and text.

Beyond making use of commands to create music and graphics, the fundamental objective for this particular assignment is to have students use good modular design in their programs. In order to have music accompanying another action and avoid excessively long procedures, the students must think carefully about how to subdivide the program. The actual student programs that result from this assignment are quite varied. One student will produce an animated picture with musical accompaniment. Another will create a sing-along by presenting text as music is played. Yet another will create an adventure story with appropriate accompanying music. No two programs are ever exactly alike.

The second problem facing the teacher is how to evaluate such varied results. Before presenting an open-ended assignment, the teacher must think carefully about the objectives to be met and how each will be evaluated. In addition, there must be clear guidelines as to what proportion of the grade will include such factors as programming style and form of output as well as what proportion will include meeting each of the objectives of the assignment. Often a check list such as the one shown below is helpful is such an evaluation.

-GRADING SHEET FOR LOGOWRITER PROJECTS USING PROCEDURES

NAME_____ PROJECT NUMBER _____ CLASS PERIOD _____

****

****

INSTRUCTIONS:

ALWAYS:

-put a procedure tree on the back of this page
-use PRINTTEXT80 to print the flip side of your LogoWriter page
-staple the listing of procedures to this sheet

IF YOU ARE TO TURN IN A RUN:

-run the program to demonstrate that it works
-staple a print of the front of your LogoWriter page to this sheet

IF YOU ARE TO PUT YOUR PROGRAM ON DISK, fill in each blank below:

DISK NAME: _____
PAGE NAME: _____
PROGRAM NAME: _____

****

****

STYLE (35%)

Meaningful procedure/input/variable names----------
(5)_____%

Procedures not chained; program uses only user defined proc. calls--
(5)_____%

One statement per line; no unnecessary rewriting of code----------
(5)_____%

Procedures/lines of reasonable length: not too short; not too long-
(5)_____%

Procedures well formatted----------
(5)_____%

Uses good overall programming style----------
(10)_____%

PROBLEM SOLUTION (25%)

Procedure tree included on the back; procedure tree correct---------
(5)_____%

Flip side contains only procedures used in the program----------
(5)_____%

Logic used in solution is straightforward/can be generalized--------
(5)_____%

Modularity; procedure names reflect procedure functions----------
(10)_____%

COMPLETE AND CORRECT RESULTS (25%)----------
(25)_____%

(Program produces the results required by the project statement)

OUTPUT (15%)

Good use of page; correctly formatted; easy to read----------
(5)_____%

User friendly; easy to understand purpose of program----------
(5)_____%
Front of page blank or contains appropriate information----------
(5)_____%

EXTRA CREDIT (20%)
Embellishments (above and beyond minimum)----------
(5)_____%
Approved extra enhancement----------
(15)_____%

****
COMMENTS:

Total percent =

_____%
___See the flip side
of your page (on times maximum possible points = * _____ =
_____points
your disk) for
comments. less days late * 5 = _____ * 5 = -
_____points

___See your attached pages Final score (out of _____) =
_____points
for comments.
___See above for comments

Results

The most obvious result of this change in type of assignments is that students become more engaged in their projects. No longer is the computer programming assignment just an exercise to complete; rather it becomes more personalized. Students take great pride in their accomplishments when they can inject their own personal touches into their work.

What is initially less obvious is the ways in which working with others changes. In a traditional environment, the most able students are quite "popular" just before an assignment is due. Students who rely on their more capable peers in this way, simply do not learn much by solving any particular problem. When students generate their own projects, there is no longer a tendency to ask others for "the answer". Instead, the quality of interactions changes. Students share techniques, ideas, and help each other debug projects, in the process often learning more than just completing only their own work.

Evaluation of these open-ended projects is certainly more interesting for the teacher. One never tires of grading the same code over and over, and the creativity of students generates constant surprises. However, the process of evaluation is quite different. While considerations such as programming style and the use of modularity in a program likely remain much the same, simply checking the results of a data set is no longer possible. Each project must be examined carefully to determine whether the objectives were met. In the case of the sample problem given above, the teacher would have to run each program to see if, indeed, the music

played while text or graphics appeared dynamically on the screen. This can be a time-consuming process .

Evaluation of open ended projects often results in more success by students. Seldom does a student turn in an assignment that is "wrong". If the project does not meet the objectives set forth by the assignment, it is usually the result of a misunderstanding on the part of the student. This is often a place to apply mastery learning techniques, and allow the student to attempt the project again. Since each student's work is unique, there is no concern that the student will simply copy another's work.

The time involved in evaluating such work is well worth the effort. Knowing that students are both enjoying their work more and developing better individual skills is powerful motivation for the concerned teacher. As the instructor's skills in using this new method improve, the time involed decreases markedly. One learns how to quickly find salient features in a program and to specify problems in such a way as to minimize difficulties at the point of evaluation.

Summary

At the heart of this discussion is the question "Why do we teach programming?" Traditionally, the goals of an introductory programming course include teaching the syntax and semantics of the language, teaching control and data structures, and providing some insight into the discipline of computer science. Giving open-ended assignments in no way detracts from these goals. In fact, even more objectives can be met with the techniques described in this

paper. This approach provides for individualization, increases motivation of students, encourages creativity, discourages inappropriate assistance, encourages positive group interactions, and provides for more success experiences for students.

Further, these techniques are quite effective with students at all levels. Simply telling more advanced students that they are to implement a stack or linked list in a meaningful way requires students to practice a new technique, but allows them room for individual expression.

In a time when our overall goal is to have students be comfortable with the computer; to make the technology a part of their lives, it seems far more important to encourage students to use the computer creatively rather to insist that students march lock-step through traditional and often meaningless exercises. The result of such changes would likely be a generation of students more comfortable with the computer as an integral and useful part of their world.

References

Amsbury, Wayne, *Structured Basic and Beyond,* Computer Science Press, Potomic Maryland, 1980.

Conway, Richard, Gries, David, and Zimmerman, E. C., *A Primer on Pascal,*, Winthrop, Cambridge, Massachusetts, 1978

Cooper, Doug and Clancey, Michael, *Oh! Pascal,*, W.W.Norton, New York, 1982.

Papert, Seymour, *Mindstorms: Children, Computers, and Powerful Ideas,*
Basic Books, New York, 1980.